PA 01-308

reference 3

⑲日本国特許庁(JP)　⑪特許出願公開

⑫公開特許公報(A)　平3-126169

⑤Int.Cl.⁵	識別記号	庁内整理番号	⑬公開　平成3年(1991)5月29日
G 06 F 15/40	520 A	7218-5B	
12/00	301 P	8944-5B	
15/16	380 Z	6945-5B	

審査請求　有　請求項の数 10 (全17頁)

⑭発明の名称　データベース・リレーションの結合方法

⑳特　願　平2-217257

㉒出　願　平2(1990)8月20日

優先権主張　㉜1989年10月5日㉝米国(US)㉞417366

㉑発明者　ダニエル・マニュエル・デイアズ　アメリカ合衆国ニューヨーク州マホパツク、パイク・プレース16番地

㉑発明者　ジヨエル・レオナード・ウオルフ　アメリカ合衆国ニューヨーク州ゴールデン・ブリツジ、チエロキー・コート(番地なし)

㉑発明者　フイリツプ・シーラング・ユ　アメリカ合衆国ニューヨーク州チヤパツカ、ストノーワエ18番地

㉑出願人　インターナシヨナル・ビジネス・マシーンズ・コーポレーション　アメリカ合衆国10504、ニューヨーク州　アーモンク(番地なし)

㉔代理人　弁理士　頓宮　孝一　外1名

明　細　書

1．発明の名称　データベース・リレーションの結合方法

2．特許請求の範囲

(1) P個のプロセッサを有する並列リレーショナル・データベース環境内で共通フィールド上で2つのデータベース・リレーションを結合する方法であって、

(a) 共通フィールド内の各値が第1のリレーションのパーティションの1つに一義的に対応し、かつ第2のリレーションのパーティションのそれと対応する1つに一義的に対応するように、第1のリレーションを複数のパーティションに区分しかつ第2のリレーションをそれらに対応する複数のパーティションに区分し、それにより、それぞれが第1のリレーションのパーティションを第2のリレーションの対応するパーティションと結合するタスクから構成される、ジョブの現結合を定義して、それらのジョブが1つのプロセッサに適合するように、各ジョブの実行時間を全実行時間の1／P以下に短縮するステップと、

(b) 最小メークスパン最適化技法を使用して、P個のプロセッサ間でジョブの現結合をスケジューリングするステップと、

(c) P個のプロセッサが、ステップ(b)でスケジュールされたジョブの現結合を実行する場合に生ずるはずの完了時間スキュー量を推定するステップと、

(d) 推定スキュー量を標準スキュー量と比較するステップと、

(e) 推定スキュー量が標準スキュー量に適合する場合は、最後にスケジューリングされたジョブの現結合をP個のプロセッサを使って実行し、それにより前記2つのデータベース・リレーションを共通フィールド上で最小の完了時間スキューで結合するステップと

を含む方法。

(2) 推定スキュー量が標準スキュー量に適合しない場合は、さらに

(f) 第1及び第2のリレーションの最大パーティションの1つを少なくとも2つのより小さなパーティションで置き換え、それにより、置き換えられたパーティションによって定義されるジョブについて、それぞれが第1のリレーションのより小さい置換パーティションの1つを第2のリレーションの対応するより小さい置換パーティションと結合するタスクから構成される、置換ジョブを定義するステップと、

(g) 置き換えられたジョブの推定実行時間を、置換ジョブの推定実行時間で置き換えることにより、推定実行時間の現集合を更新するステップと、

(h) 完了時間スキュー推定値が最小になるように、P個のプロセッサの間でのジョブの現集合のスケジューリングを更新するステップと、

(i) P個のプロセッサが、ステップ(h)でスケジューリングされたジョブの現集合を実行する場合に生ずるはずの完了時間スキュー量を推定するステップと、

(j) ステップ(i)で推定された完了時間スキュー量を標準スキュー量と比較するステップと、

(k) ステップ(j)でスキュー量が標準スキュー量に適合するか、またはすべてのパーティションが所定の最大限度まで再区分されてしまうまで、ステップ(e)ないし(j)を繰り返し実行するステップと

を含む、請求項1に記載の方法。

(3) 区分ステップがさらに、

(l) 単一プロセッサが前記ジョブのそれぞれを実行するのにかかる時間を推定し、前記推定値が推定実行時間の現集合を形成するステップと、

(m) その推定実行時間が推定実行時間の合計をPで割った値より大きいジョブに対応するパーティションを再区分するステップと、

(n) 置き換えられたジョブの推定実行時間を、置換ジョブの推定実行時間で置き換えることにより、推定実行時間の現集合を更新するステップと、

(o) 推定実行時間の合計をPで割った値より大きな推定実行時間が、ジョブの現集合内のどのジョブについてもなくなるまで、ステップ(m)及び(n)を繰り返し実行するステップと

を含む、請求項2に記載の方法。

(4) 再区分ステップがさらに、

(p) 共通フィールド内のそれぞれの値が第1のリレーションのより小さいパーティションの1つに一義的に対応し、かつ第2のリレーションのより小さいパーティションの対応する1つに一義的に対応するように、複数の明確な値を含む第1のリレーションのパーティション、及び第2のリレーションのそれと対応するパーティションを、少なくとも2つのより小さいパーティションで置き換え、それにより、ジョブの現集合内のステップ(p)の任意ジョブについて、それぞれが第1のリレーションのそのようなより小さいパーティションの1つを第2のリレーションの対応するより小さいパーティションと結合するタスクから構成される、置換ジョブを定義するステップと、

(q) 第1及び第2のリレーションの対応するパーティションが共通フィールド内にただ1つの明確な値を含むというジョブに関して、推定実行時間に分割したとき、推定実行時間の合計をPで割った値より小さい時間を与える最小の整数をXとして、そのような対応するパーティションのより大きいものをX個のより小さいパーティションで置き換えて、それにより、ジョブの現集合内のそのようなステップ(q)のジョブについて、それぞれが一方のリレーションのX個のより小さいパーティションの1つを他方のリレーションの未変更の対応するパーティションと結合するタスクから構成される、置換ジョブを定義するステップと

を含む、請求項3に記載の方法。

(5) V_1及びV_2が前記第1及び第2のリレーションの結合カラムのドメイン内の2つの値であり、$V_1 < V_2$を満たす対(V_1, V_2)がタイプ1の対として定義され、$V_1 = V_2$を満たす対(V_1, V_2)がタイプ2の対として定義され、さらにステップ(f)の前に、

(r) 少なくとも2つのより小さいパーティションで置き換えるために、最大のタイプ1の対を選択するステップと、

(n) タイプ1の対がない場合には、続いて、P個のプロセッサを使って最後にスケジューリングされたジョブの現組合を実行し、それにより、前記の2つのデータベース・リレーションを共通フィールド上で最小の完了時間スキュー量で結合するステップと

を含む、請求項2に記載の方法。

(6) V_1及びV_2が前記第1および第2のリレーションの結合カラムのドメイン内の2つの値であり、$V_1 < V_2$を満たす対（V_1, V_2）がタイプ1の対として定義され、$V_1 = V_2$を満たす対（V_1, V_2）がタイプ2の対として定義され、ステップ（m）で前記ジョブの1つの推定時間が合計実行時間の1／Pより大きいときは、さらに

(t) 前記の1つのジョブがタイプ1の対であるかどうか判定するステップと、

(u) タイプ1の対である場合には、それを、1つがタイプ2である少なくとも2つのパーティションで置き換えるステップと、

を含む、請求項3に記載の方法。

(7) 前記の1つのジョブがタイプ2の対である場合には、次いで

(v) タイプ2の対の多重度を増加できるかどうかを判定するステップと、

(w) その多重度を増加できる場合には、最小の多重度を見つけ、ジョブの現組合を改訂するステップと、

を実行する、請求項6に記載の方法。

(8) タイプ2の対の多重度を増加できない場合には、

(x) タイプ2の対を多重度Pをもつように割り当てるステップと、

(y) P個のジョブのそれぞれを異なるプロセッサに割り当てるステップと、

を実行する、請求項7に記載の方法。

(9) ステップ（a）が、2つのハッシュ関数H_1及びH_2を使用する階層的二重ハッシュ処理技法によって実行され、その際に、H_2は、H_1によって作成された各ハッシュ・パーティションをさらにより小さなパーティションに分割するために使用される、請求項1に記載の方法。

(10) 結合操作においてステップ（e）の間に前記プロセッサの進行状況を監視し、結合操作の進行が、所定のしきい値を超えてバランスを失った場合に、ステップ（b）ないし（d）を反復するという、請求項1に記載の方法。

3．発明の詳細な説明

A．産業上の利用分野

本発明は、一般に、マルチプロセッサ環境でのリレーショナル・データベースの管理に関し、より具体的には、結合操作を別々のジョブに区分し、それらのジョブを複数のプロセッサ間で最適にスケジューリングすることにより、データ・スキューの存在下で並列リレーショナル・データベース環境にある共通フィールド上の2つのデータベース・リレーションを結合することに関する。

B．従来の技術

各種リレーショナル・データベース・システムにおける共通の操作は、共通ドメインで定義されたそれぞれのカラム上の2つのリレーションの自然結合である。たとえば、C．デート（Date）著、"An Introduction to Database Systems"、Vol．1、第3版、Addison-Wesley社刊（1982年）のpp．209～210にある自然結合の説明を参照されたい。結合の結果、各行が、元のそれぞれのリレーションから1行ずつ、共に当該の結合カラム内で同じ値をもつ2つの行の連結である、新しいリレーションが得られる。

2つのリレーションの結合を計算するためのよく使用されるアルゴリズムは、M．ブラスゲン（Blasgen）及びK．エスワラン（Eswaran）の論文 "Storage and Access in Relational Databases"、IBM Systems Journal、Vol．4、pp．363以降（1977年）に記載されているようなソート／マージ技法である。それは、次のように簡潔に要約することができる。まず（必要があれば）それぞれのリレーションが結合カラムに従ってソートされる。第2に、2つのソートされたリレーションが、明白なインタロック・シーケンスで走査され、等しい値をもつ行についてマー

リされる。

ソート／マージ結合が、マルチプロセッサ・データベース・システム上で並列に実行されるときは、リレーションの結合カラム内にデータ・スキューが生じるという問題がある。一般的に、スキューの問題は、上記文献に記載された結合アルゴリズムによっては解決されない。並列ソート／マージ結合に関する初期の論文には、D．ビトン（Bitton）、H．ボラル（Boral）、D.J.デウィット（DeWitt）、W.K.ウィルキンソン（Wilkinson）の論文 "Parallel Algorithms for the Execution of Relational Database Operations"、ACM Trans. on Database Systems、Ｖｏｌ．８、Ｎｏ．３、１９８３年９月、ｐｐ．３２４～３５３がある。上記論文では、２つの外部並列ソート・アルゴリズムが提案され、それらは並列２進マージ及びブロック・ビットニック・ソートと呼ばれている。どちらのアルゴリズムでも、ソートされたランをディスクに書き込み、ツー・ウェイ・マージを使って、ディスクからランをマージする。マージ・フェーズが、異なるプロセッサにマップされ、最終マージはシーケンシャルとなる。

P．バルドゥリエス（Valduriez）及びG．ガルダリン（Gardarin）の論文 "Join and Semijoin Algorithms for a Multiprocessor Database Machine"、ACM Trans. on Database Machines、Ｖｏｌ．９、Ｎｏ．１、１９８４年３月、ｐｐ．１３３～１６１には、ｋウェイ・マージに一般化されたアルゴリズムが記載されている。その結果、単１プロセッサ上でシーケンシャルにマージされるｐ個（ｐ＜ｋと仮定する）のリストが得られる。

Ｊ．Ｐ．リチャードソン（Richardson）、Ｈ．ルー（Lu）、Ｋ．ミッキリネニ（Mikkilineni）の論文 "Design and Evaluation of Parallel Pipelined Join Algorithms"、ACM SIGMOD 1987、サンフランシスコ、１９８７年５月、ｐｐ．１６０～１６９には、マージ／結合操作を並列化する方法が記載されている。この方法では、リレーションT_1及びT_2が、m_1及びm_2個のランにマージされる。T_1が大きい方のリレーションであると仮定すると、T_1の各ランが１つのプロセッサに割り当てられる。各プロセッサは、T_2のm_2個のランをマージし（すなわち、T_2の最終マージが、少なくともプロセッサの数と同じ回数だけ繰り返される）そのプロセッサに割り当てられたT_1のランとマージ／結合する。この方法は、結合される２つのリレーションの射影の一方がきわめて小さいようなときには、うまく働く。上記論文には、１つのリレーションが小さい場合に有用な別のアルゴリズムも記載されている。

S.G.アクル（Akl）及びN．サントロ（Santoro）の論文 "Optimal Parallel Merging and Sorting Without Memory Conflicts"、IEEE Trans. on Comp.、Ｖｏｌ．Ｃ－３６、Ｎｏ．１１、１９８７年１１月、ｐｐ．１３６７～１３６９には、２つのソートされたリストを、各リストを区分することにより並列にマージすることが考察されている。

２つのリレーションの結合を計算するための別のよく使用されるアルゴリズムは、Ｄ．Ｊ．デウィット、Ｒ．Ｈ．ガーバー（Gerber）、Ｇ．グレーフェ（Graefe）、Ｍ．Ｌ．ヘイテンス（Heytens）、Ｋ．Ｂ．クマル（Kumar）、Ｍ．マラリクリシュナ（Muralikrishna）の論文 "Multiprocessor Hash-based Join Algorithms"、Proc. 11th VLDB（１９８５年）に記載されているハッシュ結合技法である。マルチプロセッサ・システムに関しては、ハッシュ結合技法は次のように簡潔に要約できる。まず、両方のリレーションを、結合カラムに応じて（必要があれば）ハッシュ・パーティションにハッシュする。ハッシュ・パーティションの数は、一般に、プロセッサの数に等しくセットされる。次に、ハッシュ・パーティションを、２つのリレーションの対応するパーティションが同一プロセッサ上に存在するように、プロセッサ間に分配する。第２に、２つのリレーションの対応するハッシュ・パーティションを１つに結合する。

結合順会の性能は、マルチプロセッサの使用に

よってスピードアップできることもあるが、この種の従来型結合アルゴリズムによるスピードアップは、M．S．ラクシュミー（Lakshmi）及びP．S．ユー（Yu）の論文 "Effect of Skew on Join Performance in Parallel Architectures"、Proc. Intl. Symposium on Databases in Parallel and Distributed Database Systems（１９８８年）に記載されているように、データ・スキューが存在する場合はきわめて限られている。D．A．シュナイダー（Schneider）及びD．J．デウィットの論文 "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment"、Proc. ACM Sigmod Conference（１９８９年）では、４つの並列結合アルゴリズムの性能を評価している。上記論文では、データ・スキューが高い場合には、ハッシュ結合以外のアルゴリズムを考える必要があると示唆している。R．C．フー（Hu）及びR．R．ムンツ（Muntz）の論文 "Removing Skew Effect in Join Operation on Parallel Processors"、Technical Report CSD-890027、UCLA（１９８９年）には、最大スキュー要素を特定し、それを処理するために複数のプロセッサを割り当てる単純なハッシュ結合アルゴリズムを提案している。

C．発明が解決しようとする課題

したがって、本発明の目的は、ソート／マージ技法、及びハッシュ結合技法によって、並列リレーショナル・データベース環境において２つのリレーションの自然結合のための全実行時間を最小にする効率的な技法を提供することである。

本発明の別の目的は、２つのリレーションの結合カラム内に存在しうるデータ・スキューを効率的に処理する、マルチプロセッサ・データベース・マシン上で動作するソート／マージ技法及びハッシュ結合技法を提供することである。

D．課題を解決するための手段

本発明の広義の教示によれば、結合操作は３段階で実行されるが、任意選択として第４段階を伴うこともある。第１段階は準備段階であり、その詳細は使用する基礎的結合アルゴリズムに応じて変わってくる。この準備段階は、前処理を行ない、その結果は、最終結合操作用のサブタスクを定義するための基礎として以降の段階で使用される。第１段階で提供されたデータを第２段階で使ってサブタスクを定義し、データ・スキューが存在する場合でも最終結合操作での各プロセッサの負荷がほぼ等しくなるように、これらのサブタスクを異なるプロセッサに最適に割り当てる。この第２段階は、本発明にとって最重要な割当て段階である。第２段階の詳細は、基礎的結合アルゴリズムが異なれば異なってくる。しかし、第２段階で、サブタスクを定義し割り当てる一般的方法は、基礎となるアルゴリズムが異なっても類似している。第２段階でサブタスクの定義及び割当てを完了すると、処理のための割当て、及び第３段階における２つのリレーションの最終的結合に応じて、データがプロセッサの間で発送される。最終結合操作でどんな方法を使用するかは、使用する基礎的結合アルゴリズムによって決まる。任意選択として、第３段階で実行される実際の結合中に結合操作のバランスがくずれた場合には、サブタスクの動的再割当てがありうる。

前述したように、結合操作は、異なる基礎的結合方法を使用して実施できる。本発明の第１の態様では、多重プロセッサ・データベース・システム上での並列なソート／マージ結合に基づく方法を記述する。第１段階では、第１及び第２のリレーションが、プロセッサの数に対応する数の集合に区分され、結合されるカラム上でソートされる。第１段階からのこのソートされたデータが、第２段階に提供される。第２段階では、第１段階からのソートされたデータが、サブタスクを定義するため様々なレンジ及び多重度に再区分され、各サブタスクは１つのレンジのパーティションからのデータを結合する。１つのプロセッサが各サブタスクを実行するのに要する時間を推定し、数個のプロセッサ間のソート操作のバランスをとるために必要なだけパーティションをさらに分割する。最小メークスパン最適化技法に従って、ジョブを

プロセッサ間でスケジューリングする。このアルゴリズムは、プロセッサ間でのジョブのスケジューリングを推定スキューに基づいて更新することにより、２つのリレーションの結合カラム内に存在しうるデータ・スキューを効率的に処理する。

本発明の第２の態様では、マルチプロセッサ・データベース・システム上での並列なハッシュ結合アルゴリズムに基づく方法を記述する。この技法は、データ・スキューを処理するために特別に設計されている。ここに提案するアルゴリズムは、階層ハッシュ処理のコンセプトに基づいている。階層ハッシュ処理を最小メークスパン最適化アルゴリズムと組み合わせて、ハッシュ・パーティションを繰り返し分割し、また複数のプロセッサ内で均等にハッシュ・パーティションの割当てを行なう。この解析の一環として、あるパーティションのサイズによって、負荷のバランスが望ましい状態より悪化したとき、区分操作を改善する。この評価段階中は、実際のハッシュ処理は実行されない。その代りに、最適のハッシュ処理が後に実行できるように、各レベルでのハッシュ処理の有益性を評価する。

B．実施例

以下の説明では、話を簡単にするために、マルチプロセッサ・データベース・マシン内の各プロセッサは同じ計算能力をもつと仮定する。その他の点での並列データベース・アーキテクチャの性質は、本発明にとって余り重要ではない。図面、特に第１図を参照すると、マルチプロセッサ・データベース・マシンの一般的構成のブロック・ダイヤグラムが図示されている。Ｐ個のプロセッサ１０$_1$ないし１０$_P$が、インタコネクト・ネットワーク１２を介して互いにリンクされている。これらのプロセッサは、インタコネクト・ネットワーク１２を介してデータを交換する。各種プロセッサをインタコネクトするためにどんな方法を使用するかは、その方法が当面のタスクに対して十分な帯域幅をもっているかぎり重要ではない。このデータベース・マシンはまた、２次記憶のために通常の磁気ディスク装置１６$_{1D_1}$ないし１６$_{PD_P}$を使用する。ソートまたは結合されるリレーションは、これらの磁気ディスク装置上に記憶される。

結合属性がとる値の分布は、結合操作の実行時間に対して相当な影響をもつ。多くのデータベースでは、１つの属性に対してある値が他の値より頻繁に発生し、その結果スキューのある分布が生じる。第２図に示した３つのリレーション、ＣＵＳＴＯＭＥＲＳ、ＯＲＤＥＲＳ、ＳＵＰＰＬＩＥＲＳを考えてみる。ＣＵＳＴＯＭＥＲＳリレーション上のカストマＩＤは一般的な値をとるが、ＯＲＤＥＲＳリレーション上の同じ属性は非一般的な値をとることができる。というのは、カストマが複数の品目を注文する場合があるからである。さらに、所与の品目を複数のカストマが注文することがあり、複数の供給者が１つの品目を供給することもありうる。

その注文が未処理のすべてのカストマをリストするために必要な照合があると、ＯＲＤＥＲＳリレーションとＣＵＳＴＯＭＥＲＳリレーションがカストマＩＤカラム上で結合されて、単一のスキューをもつ結合を生ずる。未処理の注文を満たすことのできるすべての供給者をリストするために必要な照合が行なわれると、ＯＲＤＥＲＳリレーションとＳＵＰＰＬＩＥＲＳリレーションが品目カラム上で結合されて、二重スキューをもつ結合を生ずる。単一スキュー及び二重スキューという用語は、一方または両方のリレーションの結合属性がスキューのある分布をもつことを表すために使用する。

本発明による結合操作は、第３図に示したように、３つの段階と、任意選択として第４の段階をもつ。段階１は準備段階であり、本発明のすべての態様に共通である。ただし、この段階の詳細は、使用する基礎的結合アルゴリズムに応じて変わってくる。この段階については、２つの基礎的結合アルゴリズムに関してより詳細に説明する。具体的には、まず本発明のソート／マージに基づく実施例に関して段階１を説明し、次に、本発明のハッシュに基づく実施例に関して段階１を説明する。

段階１の準備フェーズでは前処理を行ない、そ

の結果を、段階3の最終結合操作用のサブタスク定義のための基礎として段階2で使用する。より具体的には、段階1で提供されるデータを使ってサブタスクを定義し、データ・スキューが存在する場合でも段階3の最終結合操作でプロセッサの負荷がほぼ等しくなるように、これらのサブタスクを異なるプロセッサに最適に割り振る。このように、段階2は割当て段階であり、本発明にとって最も重要なものである。以下の説明では、ソート／マージ、及び本発明のハッシュに基づく2つの実施例に関して、段階2に含まれる諸ステップを詳細に説明する。

段階2でサブタスクの定義及び割当てが完了すると、結合されるリレーションのパーティションが、段階3で2つのリレーションを処理し最終的に結合するために、割り当てられたプロセッサに発送される。最終的結合操作でどんな方法を使用するかは、基礎的結合方法に応じて決まる。任意選択として、段階4は、プロセッサがその進行状況を報告するものである。段階2での割当ては、サブタスク時間の推定値に基づくので、段階3で開始される結合操作のバランスがくずれることがありうる。このアンバランスが所定のしきい値を超えた場合、段階2の割当てを、任意選択として動的に変更することもできる。

これから説明する本発明の第1の態様は、ソート／マージ結合アルゴリズムに基づく基礎的結合方法に関するものである。リレーションR_1がN_1行を含み、リレーションR_2がN_2行を含むと仮定する。Pはデータベース・システム内のプロセッサの数を表す。

本発明の第1段階では、R_1の1組の行を、できるだけ均等に、1組がほぼN_1/P行からなるP組に区分する。言いかえると、パーティションは、1組がそれぞれ$\lceil N_1/P \rceil$行からなる$N_1 - P\lceil N_1/P \rceil$個の集合と、1組がそれぞれ$\lfloor N_1/P \rfloor$行からなる$P - N_1 + P\lceil N_1/P \rceil$個の集合をもつ。ここで$\lceil x \rceil$は、実数xより大きいかまたは等しい最小の整数を表し、$\lfloor x \rfloor$は、x以下の最大の整数を表す。次に各プロセッサに、ソートすべきパーティションの1つを与える。各プロセッサは、ほぼ等しい時間で、各自のタスクを完了しなければならない。第2のリレーションも同様にソートする。この段階の終了時には、P個のソートされた「ラン」が2組ある。

本発明の第2段階を導入するために、結合カラムのドメイン内の2つの値をV_1, V_2 ($V_1 \leq V_2$)とする。2P個のソートされたランのどれか1つ、たとえばリレーション$I \in \{1, 2\}$とプロセッサ$J \in \{1, \ldots, P\}$に対応するランが与えられているものとすると、区間$[V_1, V_2]$内にソート・カラム値をもつすべての行からなる明確に定義された（恐らく空の）連続する部分集合ρ_{I,J,v_1,v_2}が存在する。ρ_{I,J,v_1,v_2}のそれぞれを最終ソート及びマージのために単一プロセッサに発送すると、結合操作の関りの部分全体の独立サブタスクτ'_{v_1,v_2}が得られる。ここで上付き文字は、単一のプロセッサが関与していることを強調するためのものである。このことの意味は、以下の説明で明らかになる。このサブタスクを実行するのにかかる時間は$T'_{v_1,v_2} = A(I_1 + I_2) + BO$と推定される。ここで

$$I_I = \sum_{J=1}^{P} \mathrm{card}(\rho_{I,J,v_1,v_2})$$

はR_Iからの入力の行数、OはR_1とR_2のマージからの出力の行数、AとBは既知の定数である。各ρ_{I,J,v_1,v_2}の値が、V_1とV_2の間の基礎ドメイン内のD_{v_1,v_2}個の要素にわたって均等に分配されていると仮定すると、$O = I_1 I_2 / D_{v_1,v_2}$を計算することができる。

$V_1 < V_2$という特別の場合には、Oの計算は単に推定値になる。$V_1 < V_2$を満たす対(V_1, V_2)をタイプ1の対と呼ぶ。$V_1 = V_2$という特別の場合には、Oの式は$O = I_1 I_2$に漸近される。事実、出力は2つの入力のちょうど外積であり、したがって、この式は、この場合厳密に正確である。$V_1 = V_2$を満たす対(V_1, V_2)をタイプ2の対と呼ぶ。実際に、タイプ2の対では、$V = V_1 = V_2$とすると、さらに、2つの集合

$\bigcup_{j=1}^{p} \rho_{1,j,v_1,v_2}$ と $\bigcup_{j=1}^{p} \rho_{2,j,v_1,v_2}$

の大きい方をできるだけ均等にM個（ただし、1≦M≦P）の行の集合に区分し、さらに小さい独立サブタスク$\tau^1_{v_1,v_2}, \ldots, \tau^M_{v_1,v_2}$を作成できる可能性を考えることができる。サブタスク$\tau^m_{v_1,v_2}$, m ∈ {1, ..., M}では、小さい方の集合と大きい方の集合のm番目のパーティションとの外積が単一プロセッサ上で実行される。たとえば、第1のリレーションの方が大きい行数を含むと仮定すると、このサブタスクを実行するのにかかる時間は、$T^m_{v_1,v_2} = A(I^m_1 + I_2) + BI^m_1 I_2$となる。ここで、$I^m_1$はm番目のパーティション内の行数である。M個のサブタスクをそれぞれ異なるプロセッサで実行すべきであるとは主張しないが、実際にはそうなる可能性が大きい。明らかに、M＞1個のサブタスクを実行するのは、1つのサブタスクを実行するよりも効率が低い。というのは、小さい方のリレーション（この場合にはR_2）からの入力を、関係する各プロセッサに発送しなければならないからである。通則なスキューを処理するために、この手法を利用しているにすぎない。タイプ2の対（V_1, V_2）は、多重度Mをもつと言う。タイプ1の対（V_1, V_2）は、多重度1をもつと言う。

次に一般的手法について述べる。結合カラムのドメイン内の対応する多重度をもつK対の値の順序シーケンスを作成するものと仮定する。このシーケンスは、次の形をもつ。

$V_{1,1} \leq V_{1,2} < \ldots < V_{k-1,1} \leq V_{k-1,2} \leq V_{k,1} \leq V_{k,2} < V_{k+1,1} \leq V_{k+1,2} < \ldots < V_{K,1} \leq V_{K,2}$

R_1とR_2の結合カラム内の各値は、区間$[V_{k,1}, V_{k,2}]$の1つ内に収まることが必要である。k ∈ {1, ..., k}について、対（$V_{k,1}$, $V_{k,2}$）の多重度をM_kで表すことにする。P個のプロセッサで実行される $\ell = \sum_{k=1}^{K} M_k$ 個のサブタスク$\tau^m_{v_{k,1},v_{k,2}}$を作成した。必要な全計算時間は、

$\sum_{k=1}^{K} \sum_{m=1}^{M_k} T^m_{v_{k,1},v_{k,2}}$と推定することができる。

この時間をプロセッサの間でできるだけ均等に分配したい。「完全」な割当ては必ずしも可能ではないが、それができれば、各プロセッサが

$\sum_{k=1}^{K} \sum_{m=1}^{M_k} T^m_{v_{k,1},v_{k,2}}/P$ 単位時間だけ使用中

となるはずである。具体的には、ジョブ全体の完了時間

$$\max_{1 \leq p \leq P} \sum_{k=1}^{K} \sum_{m=1}^{M_k} T^m_{v_{k,1},v_{k,2}} \cdot A(\tau^m_{v_{k,1},v_{k,2}}) = p$$

が最小になるように、各サブタスク$\tau^m_{v_{k,1},v_{k,2}}$をプロセッサA（$\tau^m_{v_{k,1},v_{k,2}}$）に割り当てたい。この最適化問題は、いわゆる最小メークスパン問題、またはマルチプロセッサ・スケジューリング問題である。この問題はNP完全であることが知られているが、最悪の場合の性能がかなり良く、平均の性能が優れたいくつかのきわめて高速のヒューリスティックスが存在する。具体的には、LPTヒューリスティックの説明については、R. グレアム（Graham）の論文 "Bounds on Multiprocessing Timing Anomalies"、SIAM Journal of Computing、Vol. 17、1969年、pp. 416以降、MULTIFITヒューリスティックの説明については、E. コフマン（Coffman）、M. ギャリー（Garey）、D. S. ジョンソン（Johnson）の論文 "An Application of Bin Packing to Multiprocessor Scheduling"、SIAM Journal of Computing、Vol. 7、1978年、pp. 1以降を参照されたい。

もちろん、値の対の順序シーケンス、及び対応する多重度をどのように作成するかは制御できる。本発明の第2段階の目標は、この順序シーケンスを作成するための分割統治法である。その1つの例は、第4図の流れ図に示したアルゴリズムである。

第4図を参照すると、この方法は、まず開始ブロック20から始まり、そこで、K←1に設定し、$V_{1,1}$をR_1とR_2の結合カラム内の最小値に設定し、$V_{1,2}$を最大値に設定し、かつM_1←1に設定する。さらに、そのサブタスクに対する時間を推

定し、すべての現サブタスクのリストを維持し、推定サブタスク時間の順に並べる。最初は、リスト上には1つの要素しかない。次に、判断ブロック22でテストを行なって、リスト中の最大サブタスクの推定時間がリスト中のすべてのサブタスクの推定時間の合計の1／P以下であるかどうか、言い換えると、最大サブタスクが「適合」するかどうか判定する。YESの場合は、機能ブロック24で、LPTまたはMULTIFITアルゴリズムを実行する。どちらのアルゴリズムでも、第1ステップであり、かつ計算上最も費用のかかるステップは、推定時間の順にサブタスクを並べることである。これは、この場合にはすでに済ませてある。次に、判断ブロック28でテストを行なって、このメークスパンが、完全割当ての所定の最小係数1＋εの範囲内にあるかどうか判定する。YESの場合には、処理は停止する。そうでない場合は、判断ブロック28でテストを行なって、タイプ1の対があるかどうか判定する。NOの場合は、処理は停止する。そうでない場合は、機能ブロック30で、リスト中で最大のタイプ1の対を選択する。次に、機能ブロック32で、最大のタイプ1の対（$V_{a,1}$, $V_{a,2}$）を、次のように2ないし3つの部分に分割する。R_1には、P個のソートされた行$\rho_{1,j,v_{a,1},v_{a,2}}$からなる合計$I_1$個の要素がある。$R_2$には、P個のソートされた行$\rho_{2,j,v_{a,1},v_{a,2}}$からなる合計$I_2$個の要素がある。合計で、合計$I_1+I_2$個の要素をもつ2P個のソートされた行が得られる。Z．ガリール（Galil）及びN．メジッド（Megiddo）の論文"A Fast Selection Algorithm and the Problem of Optimum Distribution of Effort"、Journal of the ACM、Vol．26、1979年、pp．58以降によるアルゴリズムで、この集合の$\left(\boxed{I_1+I_2}\right)$／2番目に大きな要素$\mu$が見つかる。これは、いわゆる選択問題の特殊な場合である。事実、このアルゴリズムは、各集合$\rho_{1,j,v_{a,1},v_{a,2}}$を3つの連続する（恐らく空の）領域、すなわち$\mu$より少ない行からなる$\rho_{1,j,v_{a,1},v_{a,2}}$と、$\mu$に等しい行からなる$\rho_{1,j,v_{a,1},v_{a,2}}$と、$\mu$より多い行からなる$\rho_{1,j,v_{a,1},v_{a,2}}$に分割する。こうして、1つのサブタスクから3つのサブタスクが作成された。第1または第3サブタスクのどちらか一方が空でもよいが、両方が空になることはない。どちらか一方がタイプ1またはタイプ2でありうる。第2のサブタスクは空にはならず、多重度1のタイプ2になる。第5図は、この方法を用いて、どのように1つの古いサブタスクからこれら3つの新しいサブタスクが作成されるかを示している。次に、Kを調整し、間隔を決定し、新しいサブタスクのそれぞれについて時間推定値を計算する。出力の行数が、いずれかのサブタスクに関して0であると計算された場合には、そのサブタスクをリストから削除することができる。2つのリレーションの一方に対応する部分は空であり、結合で何も生じない。

この時点で、判断ブロック34でテストを行なって、サブタスクのリストが空であるかどうか判定する。YESの場合は、処理は停止する。そうでない場合は、機能ブロック36でサブタスクのリストの番号を付け直し、順序を並べかえる。次に、判断ブロック22に戻る。

判断ブロック22のテストの結果がNOの場合は、判断ブロック38でさらにテストを行なって、サブタスクのリスト中の最大要素がタイプ1のものであるかどうか判定する。YESの場合は、機能ブロック32に進む。そうでない場合は、判断ブロック40でテストを行なって、各サブタスクが適合するように、リスト中の最大サブタスクに対応するタイプ2の対の多重度を新しい多重度M＜Pに増加させることが可能かどうか判定する。YESの場合は、最小のそのような多重度が見つかり、機能ブロック42で、サブタスクのリストを、この新しい多重度を組み込むように改訂する。さらに、Kを調整し、新しいサブタスクに対する推定時間を計算してから、機能ブロック36に進む。

判断ブロック40のテストの結果がNOの場合は、機能ブロック44で、サブタスクが多重度Pをもつように割り当て、P個のサブタスクのそれ

それを異なるプロセッサに割り当て、それらのサブタスクをリストから外す。さらに、Kを調整する。除外されたP個のサブタスクは、ほぼ等しい時間で完了する。次に判断ブロック34に移る。

この方法は、多重度1の対に適したものであるが、過剰なスキューを克服するため、それより高い多重度をも受け入れることができる。これらの過剰なスキューの位置は、アルゴリズムの実行中に自然にわかるはずである。推定サブタスク時間も、同様に、アルゴリズムの実行につれてますます正確になるはずである。上記テーマに関しては多数の変形があり、それらも同様に実施できる。たとえば、LPTまたはMULTIFITは、個々ブロック24を通るごとに実行する必要はない。カウンタで、最小メークスパン・アルゴリズムの実行回数を調節することができる。同様に、このアルゴリズムは、たとえ解答が完全に満足なものでなくても、別のカウンタに従ってタイム・アウトさせることができる。

第3図に示した段階3では、リレーションR₁とR₂のソートされた要素集合をディスクから読み出し、サブタスクに対応する要素集合を、割り当てられたプロセッサに発送する。次に、最終結合操作を実行するため、割り当てられたプロセッサ上でサブタスクを実行する。

任意選択として、第3図の段階4に示すように、実際の結合の進行中に、プロセッサが各自の進行状況を報告することもできる。推定サブタスク時間は、ちょうどその値なので、結合の進行がバランスを失うことがありうる。このアンバランスが所定のしきい値を超えた場合に、新しいLPTまたはMULTIFITアルゴリズムを開始することもできる。

本発明はまた、ハッシュ結合アルゴリズムに基づく方法を使用し、第3図の段階1及び2の階層ハッシュ処理を実施する二重ハッシュ処理技法を使用して実施することができる。階層ハッシュ処理技法のその他の変形については後述する。二重ハッシュ処理は、スキュー値を特定し、各プロセッサにおける結合コストのよりよい推定値をもたらし、最終的には負荷のバランスをより均等にするために採用される。

第3図の第1段階では、まずそれぞれサイズがほぼN_1/P行のR_1のほぼ等しいパーティションをもつ各プロセッサを使用する。言い換えると、パーティションは、1組がそれぞれ$\lceil N_1/P \rceil$行からなる$N_1 - P\lceil N_1/P \rceil$個の集合、及び1組がそれぞれ$\lceil N_1/P \rceil$行からなる$P - N_1 + P\lceil N_1/P \rceil$個の集合をもつ。2つのハッシュ関数を$H_1$と$H_2$とする。ここで$H_2$は、$H_1$によって作成された各ハッシュ・パーティションをより小さいパーティションにさらに分割するために使用する。H_1は、行をB_1個のハッシュ・パーティションにハッシュ処理でき、H_2は、これらのパーティションのそれぞれをB_2個のより小さいハッシュ・パーティションに細分できるものと仮定する。各プロセッサjについて、H_1のもとでその結合カラムのハッシュ値がH_1のk番目のハッシュ・パーティションに含まれ、H_2のもとでのハッシュ値がH_2のm番目のハッシュ・パーティションに含まれる行の数を$d_{k,m}(R_1)$とする。その行を、複合ハッシュ値(k, m)をもつと言う。各プロセッサは、ディスクからR_1の要素集合を読み込み、どれだけの要素結合が、H_1とH_2に基づいて細分割されたパーティションにハッシュ処理されるのかを記録するため、各プロセッサjのメイン・メモリ内にカウント$d'_{k,m}(R_1)$(k = 1, ..., B_1、及びm = 1, ..., B_2)を維持する。これらのプロセッサは、ほぼ等しい時間で、それぞれのタスクを完了する必要がある。第2のリレーションも同様にハッシュ処理されて、$d'_{k,m}(R_2)$を生ずる。この段階の終了時には、2組のカウント$d'_{k,m}(R_i)$ i = 1, 2が得られる。

第3図の第2段階の開始時に、$d'_{k,m}(R_i)$が各プロセッサにわたって加算されて、

$$\beta_{k,m}(R_i) = \sum_{1 \le j \le P} d'_{k,m}(R_i)$$

を得る。H_1からの$2PB_1$個のハッシュ・パーティションのどれか1つ、たとえばリレーションi = (1, 2)、プロセッサj = (1, ..., P)に

対応する1つのハッシュ・パーティション、及びハッシュ・パーティションk∈{1, ..., B_1}が与えられ、かつ部分集合E⊆{1, ..., B_2}が与えられているものとすると、複合ハッシュ値(k, m)、m∈Eをもつすべての行からなる明確に定義された(恐らく空の)部分集合$\rho_{1,k,m}$が存在する。$\rho_{1,k,m}$のそれぞれを最終結合のために単一プロセッサに発送すると、結合操作の残りの部分全体の独立サブタスク$\tau^1_{k,E}$が得られる。ここで上付き文字は、単一のプロセッサが割当てていることを強調するためのものである。このことの意味は間もなく明らかになる。このサブタスクを実行するのにかかる時間は、$T^1_{k,E}=A(I_1+I_2)+BO$と推定される。ここで

$$I_1=\sum_{m\in E}\beta_{k,m}(R_1)$$

はR_1からの入力の行数、OはR_1とR_2のマージからの出力の行数、AとBは既知の定数である。各$\rho_{1,k,m}$、m∈Eの値が、複合ハッシュ値(k, m)をもつ基礎ドメイン内の$D_{k,m}$個の要素にわたって均等に分配されていると仮定できる場合には、

$$O=\sum_{m\in E}\beta_{k,m}(R_1)\,\beta_{k,m}(R_2)/D_{k,m}$$

を計算することができる。

Eが単一の値、たとえばmを含む場合には、さらに次の2組

$$\bigcup_{j=1}^{J}\rho_{1,j,k,(m)}\quad\text{と}\quad\bigcup_{j=1}^{J}\rho_{2,j,k,(m)}$$

の大きい方をできるだけ均等にM個(1≦M≦P)の行の集合に区分し、さらに小さい独立サブタスク$\tau^1_{k,(m)},\ \ldots,\ \tau^M_{k,(m)}$を作成できる可能性を考えることができる。サブタスク$\tau^j_{k,(m)}$, j∈{1, ..., M}では、小さい方の集合と大きい方の集合のj番目のサブパーティションとの結合が、単一プロセッサ上で実行される。たとえば、第1のリレーションの方が大きい行数を含むと仮定すると、このサブタスクを実行するのにかかる時間は、$T^j_{k,(m)}=A((I_1/M)+I_2)+BI_1I_2/M$となる。M個のサブタスクのそれぞれを異なるプロセッサ上で実行すべきであるとは主張しないが、実際にはそうなる可能性が大きい。明らかに、M>1個のサブタスクを実行するのは、1つのサブタスクを実行するよりも効率が悪い。というのは、小さい方のリレーション(この場合にはR_2)からの入力を、関係する各プロセッサに発送しなければならないからである。過剰なスキューを処理するために、この手法を利用しているにすぎない。複合ハッシュ・パーティション(k, {m})は、多重度Mをもつと言う。そうでない場合は、多重度1をもつと言う。

次に、一般的手法について述べる。($V_{k,1}$, $V_{k,2}$)の形のk対の順序シーケンスを作成するものと仮定する。ここで、$V_{k,1}$∈{1, ..., B_1}、$V_{k,2}$⊆{1, ..., B_2}。$V_{k,2}$がただ1つの要素を含む場合は、($V_{k,1}$, $V_{k,2}$)対をタイプ2の対と呼ぶ。そうでない場合は、タイプ1の対と呼ぶ。各複合ハッシュ・クラス(i, j)について、i=$V_{k,1}$かつj∈$V_{k,2}$であるような独自のk∈{1, ..., k)が存在しなければならないという意味で、複合ハッシュ・クラスの集合を区分するためにK対のシーケンスが必要である。k∈{1, ..., k}について、対($V_{k,1}$, $V_{k,2}$)の多重度をM_kで表すことにする。P個のプロセッサで実行されるサブタスク$\tau^n_{V_{k,1},V_{k,2}}$を作成した。必要な全計算時間は、

$$\sum_{k=1}^{K}\sum_{n=1}^{M_k}T^n_{V_{k,1},V_{k,2}}\quad\text{と推定することができる。}$$

この全計算時間をプロセッサの間でできるだけ均等に分配したい。「完全な」割当ては必ずしも可能ではないが、それができれば、各プロセッサが

$$\sum_{k=1}^{K}\sum_{n=1}^{M_k}T^n_{V_{k,1},V_{k,2}}/P\quad\text{単位時間だけ使用中となるはずである。具体的には、ジョブ全体の完了時間}$$

$$\max_{1\le p\le P}\ \sum_{k=1}^{K}\sum_{n=1}^{M_k}T_{V_{k,1},V_{k,2}}\cdot\delta(\tau^n_{V_{k,1},V_{k,2}})=p$$

が最小になるように、各サブタスク$\tau^n_{V_{k,1},V_{k,2}}$

をプロセッサA（$\tau^{*}_{V_{k,1},V_{k,2}}$）に割り当てたい。この最適化問題は、いわゆる最小メークスパン問題、またはマルチプロセッサ・スケジューリング問題である。この問題はＮＰ完全であることが知られているが、最悪の場合の性能がかなり良く、平均の性能が優れたいくつかのきわめて高速のヒューリスティックスが存在する。

要は、値の対の順序シーケンス、及び対応する多重度をどのように作成するかを制御できることである。本発明の第２段階の目標は、この順序シーケンスを作成するための分割統治法である。ここでは、段階２に進むための可能な方法の例を２つ示す。

第１の方法は、グレアムによるＬＰＴヒューリスティックの変形を使用するものであり、第６図の流れ図に示されている。第６図を参照すると、まず機能ブロック５０で、集合｛（k，F），k＝1，...，B_1｝、ただしF＝｛1，...，B_2｝を、サブタスク値$T'_{k,F}$に応じて降順でソートする。すべての現サブタスクのリストを維持し、推定サブタスク時間の順に並べる。次に、判断ブロック５２でテストを行なって、リスト中の最大サブタスクの推定時間が、リスト中のすべてのサブタスクの推定時間の合計の１／Ｐ以下であるかどうか、言い換えると、最大サブタスクが「適合」するかどうか判定する。ＮＯの場合は、判断ブロック５４でテストを行なって、リスト中の最大要素がタイプ１のサブタスクであるかどうか判定する。ＹＥＳの場合は、機能ブロック５６で、最大タイプ１の対（$V_{k,1}$，$V_{k,2}$）をそれぞれ多重度が１のｃａｒｄ（$V_{k,2}$）個のタイプ２の対に分割する。次に、機能ブロック５８で、サブタスクのリストの番号を付け直し、サブタスクのリストの順序を並べかえてから、判断ブロック５２に戻る。

判断ブロック５４のテストの結果がＮＯの場合には、判断ブロック６０でさらにテストを行なって、各サブタスクが今度は適合するように、リスト中の最大サブタスクに対応するタイプ２の対の多重度を新しい多重度Ｍ＜Ｐに増加させることが可能かどうか判定する。ＹＥＳの場合は、最小のそのような多重度が見つかり、機能ブロック６２で、サブタスクのリストを、この新しい多重度を組み込むように改訂する。さらに、kを調整し、新しいサブタスクに対する推定時間を計算する。次に、機能ブロック５８に進む。

判断ブロック６０のテストがＮＯの場合には、機能ブロック６４で、サブタスクが多重度Ｐをもつように割り当て、Ｐ個のサブタスクのそれぞれを異なるプロセッサに割り当て、それらのサブタスクをリストから外す。またkを調整する。除外されたＰ個のサブタスクは、ほぼ等しい時間で完了する。次に、判断ブロック６６でテストを行なって、サブタスクのリストが空であるかどうか判定する。ＹＥＳの場合は、処理は終了する。そうでない場合は、処理は機能ブロック５８に戻る。

判断ブロック５２のテストがＹＥＳの場合は、すべてのサブタスクが適合する。機能ブロック６８で、目標メークスパン時間ＴＡＲＧＥＴをリスト中のすべてのサブタスクの推定時間の合計の１／Ｐとして計算する。このＴＡＲＧＥＴがこの場合は安定である点が重要である。以後のステップでタイプ１の対をタイプ２の対に分割することがあり得るが、推定時間の合計は変わらない。変形ＬＰＴは以下のステップで実行される。

ＴＡＲＧＥＴと、あるプロセッサにすでに割り当てられたサブタスク時間の合計の差が最大であるプロセッサＰを考える。最大サブタスク（$V_{k,1}$，$V_{k,2}$）がリスト上に残っているものと考える。判断ブロック７０でテストを行なって、それがタイプ２のサブタスクであるかどうか判定する。ＹＥＳの場合は、機能ブロック７２で、サブタスク（$V_{k,1}$，$V_{k,2}$）をプロセッサＰに割り当て、そのサブタスクをリストから外す。次に、判断ブロック８０でテストを行なって、サブタスクのリストが空であるかどうか判定する。ＹＥＳの場合は、処理は終了する。そうでない場合は、処理は判断ブロック７０に戻る。

判断ブロック７０の判定がＮＯの場合は、判断ブロック７８でテストを行なって、（$V_{k,1}$，$V_{k,2}$）がプロセッサＰに割り当てられた場合、

ＴＡＲＧＥＴとプロセッサＰに割り当てられたサブタスク時間の合計の改訂された値が負になるかどうか判定する。ＮＯの場合は、機能ブロック７２に移る。そうでない場合は、機能ブロック７４で、（$V_{k,1}$，$V_{k,2}$）がそれぞれ多重度が１ｏｃａｒｄ（$V_{k,2}$）個のタイプ２の対に分割される。機能ブロック７６で、サブタスクのリストの番号を付け直し、順序を並べかえてから、判断ブロック７０に戻る。

結合技法の段階２用の第２のハッシュ結合アルゴリズムは、第７図の流れ図に示されている。第４図の流れ図と第７図の流れ図を比較するとわかるように、ソート／マージ結合アルゴリズムとこのハッシュ結合アルゴリズムでは全体の論理は全く類似している。第７図を参照すると、処理は機能ブロック８０から始まり、まず集合｛（ｋ，Ｆ），ｋ＝１，…，B_1｝、ただしＦ＝｛１，…，B_2｝、をとり、それらを、サブタスク値$T'_{k,F}$に従って降順でソートする。すべての現サブタスクのリストを維持し、サブタスク推定時間の順に対（$V_{k,1}$，$V_{k,2}$）を次のように２つの対に分割する。各要素$m \in V_{k,2}$について、対応するサブタスク時間$T'_{V_{k,1}(m)}$が存在する。集合$\{T_{V_{k,1}(m)} \mid m \in V_{k,2}\}$に対してＬＰＴまたはＭＵＬＴＩＦＩＴを実行し、$V_{k,2}$を合計推定サブタスク時間がほぼ等しい２つの部分集合$V^1_{k,2}$と$V^2_{k,2}$に分割する。新しいサブタスクのいずれかが時間０であると推定される場合、そのサブタスクはリストから削除できる。さらに、Ｋを調整する。別法として、最大のタイプ１の対を、２と$V_{k,2}$の濃度の間の任意の数Ｑのグループに分割することもできる。この説明ではＱ＝２を選択した。判断ブロック１０４でテストを行なって、サブタスクのリストが空であるかどうか判定する。ＹＥＳの場合、処理は停止する。そうでない場合は、機能ブロック１０８でサブタスクのリストの番号を付け直し、順序を並べかえてから、制御ブロック９２に戻る。

判断ブロック９２のテストの結果がＮＯの場合は、判断ブロック１０８でテストを行なって、サブタスクのリスト中の最大要素がタイプ１のものであるかどうか判定する。ＹＥＳの場合は、機能ブロック１０２に進む。そうでない場合は、判断ブロック１１０でテストを行なって、各サブタスクが今度は適合するようにリスト中の最大サブタスクに対応するタイプ２の対の多重度を新しい多重度Ｍ＜Ｐに増加させることが可能かどうか判定する。ＹＥＳの場合は、最小のそのような多重度が見つかり、機能ブロック１１２で、サブタスクのリストを、この新しい多重度を組み込むように改訂する。Ｋを調整し、新しいサブタスクに対する推定時間を計算する。次に、機能ブロック１０６に進む。判断ブロック１１０のテストの結果がＮＯの場合は、機能ブロック１１４で、サブタスクが多重度Ｐをもつように割り当て、Ｐ個のサブタスクのそれぞれを異なるプロセッサに割り当て、それらのサブタスクをリストから外す。さらに、Ｋを調整する。除外されたＰ個のサブタスクは、ほぼ等しい時間で完了する。次に、判断ブロック１０４に進む。

この方法は、多重度１の対に適したものであるが、過剰なスキューを克服するため、それより高い多重度をも受け入れることができる。これらの過剰なスキューの位置は、アルゴリズムの実行中に自然にわかるはずである。上述の方法には多数の変形があり、それも同様に実施できる。たとえば、どちらの方法でも、反復改良アルゴリズムを終了時に適用することができる。この方法は、第８図に示されている。この図で、ＧＯＡＬは、すべてのサブタスクの推定時間の合計の１／Ｐを表している。この方法は、割り当てられたサブタスクの推定時間の合計が最大のプロセッサＰをとり出すものである。プロセッサＰで、最大のタイプ１のサブタスク（$V_{s,1}$, $V_{s,2}$）をとり出して、それぞれ多重度が１のｃａｒｄ（$V_{s,2}$）個のタイプ２の対に分割する。次に、これらのタスクを、プロセッサに再度割り当てる。第８図に示すように、最大のサブタスクが最初に再割当てされる。

第４図に示したソート／マージ結合アルゴリズムの段階２と同様に、第７図に示した第２のハッシュ結合アルゴリズムに基づく段階２用のこの方法では、ＬＰＴまたはＭＵＬＴＩＦＩＴは、機能ブロック８４を通ることに実行する必要はない。カウンタで、最小メークスパン・アルゴリズムの実行回数を制御することができる。同様に、このアルゴリズムは、たとえ解答が完全に満足なものでなくても、別のカウンタに従ってタイム・アウトさせることができる。

第３図に示した段階３では、リレーションＲ１とＲ２の要素集合をディスクから読み取り、ハッシュ関数H_1とH_2を適用し、それらの要素集合を、段階２で決定された割り当てられたプロセッサに発送する。

任意選択として、第３図に示した段階４では、実際の結合の進行中に、プロセッサが各自の進行状況を報告することもできる。推定サブタスク時間がちょうどその値なので、結合の進行がバランスを失うことがありうる。このアンバランスが所定のしきい値を超えた場合に、新しいＬＰＴまたはＭＵＬＴＩＦＩＴアルゴリズムを開始することもできる。

この階層的ハッシュ処理コンセプトを実施する方法は多数あることに留意されたい。１つの方法は、ハッシュ処理を実行する際に、段階１で出会った各個の結合カラム値上に最も頻繁に使用されるリストを保存するものである。発生頻度も維持される。結合カラム値の発生率が所定のしきい値より小さくなると、その結合カラムをリストから外す。別の方法は、ハッシュ処理を実行する前、段階１の結合カラム値のサンプリングを実行するものである。次に、最も頻繁に使用されるリストまたはサンプリング結果を使用して、スキュー分布を識別し、どのパーティションをさらに区分するべきかを案内することもできる。

要約すると、ソート／マージ・アルゴリズムまたはハッシュ結合アルゴリズムの使用に基づいて、複数のプロセッサを有する並列リレーショナル・データベース環境内で１つの共通フィールド上で２つのデータベース・リレーションを結合する方法を提供する。この方法は、第１のリレーションを多数のパーティションに区分し、第２のリレーションを対応する多数のパーティションに区分して、共通フィールド内の個々の値が、第２のリレーションのパーティションの対応する１つに一般的に対応するようにし、それにより、それぞれが第１のリレーションのあるパーティションを第２のリレーションの対応するパーティションと結合するタスクから構成される、ジョブの現集合を定義するものである。単一のプロセッサが各ジョブを完了するのに要する時間を推定し、これらの推定値が、実行推定時間の現集合を形成する。

次に、ジョブ実行時間を、推定時間の合計をプロセッサの数で割った値以下の時間に短縮することが必要である。これを行なうには、その推定実行時間が推定時間の合計をプロセッサの数で割った値より大きいジョブに対応するパーティションを再区分する。この再区分は、第１のリレーションのそのようなパーティション、及び共通フィールドに複数の異なる値を含む第２のリレーションの対応するパーティションを、少なくとも２つの

より小さいパーティションで置き換えることによって行なう。共通フィールド内の個々の値は、第1のリレーションのより小さいパーティションの1つに一意的に対応し、かつ第2のリレーションのより小さいパーティションの対応する1つに一意的に対応する。これにより、ジョブの現集合内で置換ジョブが定義される。各置換ジョブは、第1のリレーションのそのようなより小さなパーティションの1つを、第2のリレーションの対応するより小さなパーティションと結合するタスクから構成される。第1のリレーションと第2のリレーションの対応するパーティションが、共通フィールド内に1つの値のみを含むどんなジョブに関しても、そのような対応するパーティションの大きい方のものが、複数のより小さいパーティションで置き換えられ、そのようなより小さいパーティションの数は、そのようなジョブに関して、推定実行時間に分割したとき、推定実行時間の合計値をプロセッサの数で割った値より小さい時間を与える、最小の整数である。このようにして、各置換ジョブが、一方のリレーションのより小さいパーティションの1つを、他方のリレーションの未変更の対応するパーティションと結合するタスクから構成される、置換ジョブが定義される。

パーティションの再区分過程が完了すると、置き換えられたジョブの推定実行時間を置換ジョブの推定実行時間で置き換えることにより、推定実行時間の現集合が更新される。この過程は、推定実行時間の合計値をプロセッサの数で割った値より大きな推定実行時間が、ジョブの現集合内のどのジョブに対してもなくなるまで、繰り返し実行される。

本発明の重要な部分は、プロセッサ割当ての部分である。これは、最小メークスパン最適化技法を使用して、プロセッサの間でジョブの現集合をスケジューリングすることによって行なう。プロセッサがスケジュール通りにジョブの現集合を実行する場合に生ずるはずの完了時間のスキュー量を推定する。この推定値を、スキュー基準値と比較し、推定値が許容できる場合は、スケジューリングされたジョブの現集合がプロセッサ上で実行され、それにより、共通フィールド上の2つのデータベース・リレーションを最小の完了時間スキュー値で結合する。

推定スキュー値がスキュー基準値に合致しない場合は、第1のリレーション及び第2のリレーションの最大パーティションの1つが、少なくとも2つのより小さいパーティションで置き換えられて、置き換えられたパーティションによって定義されるジョブに対する置換ジョブが定義される。各置換ジョブは、第1のリレーションのより小さな置換パーティションの1つを、第2のリレーションの対応するより小さい置換パーティションと結合するタスクから構成される。次に、置き換えられたジョブの推定実行時間を置換ジョブの推定実行時間で置き換えることにより、推定実行時間の現集合が更新される。さらに、ジョブの現集合のスケジューリングが、推定完了時間スキューが最小になるように更新される。完了時間スキューの量が推定され、スキュー基準値と比較される。この過程は、スキュー量がスキュー基準値に合致するか、またはすべてのパーティションが所定の最大限度に再区分されるまで、反復して繰り返される。

F．効果

本発明は、ソート／マージ技法、及びハッシュ結合技法によって、並列リレーショナル・データベース環境において、2つのリレーションの自然結合のための全実行時間を最小にする効率的な技法を提供する。

4．図面の簡単な説明

第1図は、本発明による結合手順を実施できるマルチプロセッサ・データベース・マシンの一般構成のブロック図である。

第2図は、データ・スキューの概念を図示するために使用する3つのリレーションの例を示した図である。

第3図は、本発明による結合操作の第1段階を示すハイ・レベルのブロック図である。

第4図は、本発明によるソート／マージ結合アルゴリズムを使用する結合操作の第2段階の詳細

を示す流れ図である。

第５図は、本発明の第１の態様でサブタスクがどのようにより小さいサブタスクに分割されるかを示す図である。

第６図は、本発明による第１のハッシュ結合アルゴリズムを使用した結合操作の第２段階の詳細を示す流れ図である。

第７図は、本発明による第２のハッシュ結合アルゴリズムを使用した結合操作の第２段階の詳細を示す流れ図である。

第８Ａ図及び第８Ｂ図は、本発明のハッシュ結合操作でサブタスクがどのように再割振りされるかを示す図である。

出願人　インターナショナル・ビジネス・
マシーンズ・コーポレーション
代理人　弁理士　頓　宮　孝　一
（外１名）